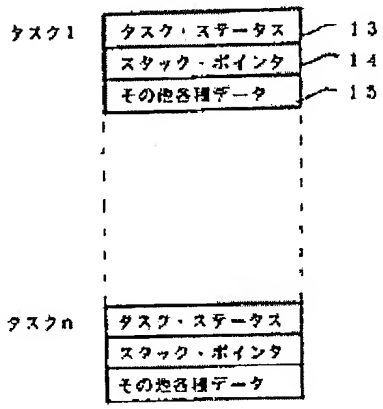
# PROGRAM PARALLEL EXECUTION DEVICE

**Patent number:** JP3078034
**Publication date:** 1991-04-03
**Inventor:** TAKAHASHI YOSHINORI
**Applicant:** YOSHINORI TAKAHASHI
**Classification:**
**- international:** G06F9/46
**- european:**
**Application number:** JP19890214667 19890821
**Priority number(s):**

Abstract of **JP3078034**

PURPOSE:To facilitate the parallel processing of a single-task operating system by saving registers in the dedicated area of a task when a program being executed is interrupted at a request from the task.

CONSTITUTION:When the program being executed is interrupted at the request from the task 1, all registers of a task control table except the stack pointer 14 of the task 1 are saved in the stack area. Then the pointer register is saved in the stack pointer of another task of the control table. At this time, when the task 1 waits for an event, the corresponding task status 13 of the control table is disabled and necessary data are set to various other data 15. When the task 1 is executed, the pointer 14 of the control table is reloaded in a register and then other registers in the stack area are reloaded. Consequently, the parallel processing of the single-task operating system is facilitated.



⑲日本国特許庁(JP)　⑪特許出願公開

⑫公開特許公報(A)　平3-78034

| ⑤Int. Cl.⁵ | 識別記号 | 庁内整理番号 | ㊸公開　平成3年(1991)4月3日 |
|---|---|---|---|
| G 06 F 9/46 | 340 B | 8945-5B | |

審査請求　未請求　請求項の数 1　(全4頁)

㊹発明の名称　プログラム並行実行装置

㉑特　願　平1-214667

㉒出　願　平1(1989)8月21日

㉒発明者　高橋　義則　京都府長岡京市下海印寺西条6

㉑出願人　高橋　義則　京都府長岡京市下海印寺西条6

明細書

1．発明の名称

プログラム並行実行装置

2．特許請求の範囲

コンピュータ装置において、

実行中のプログラムを中断する手段、

コンピュータ装置の内部または外部から自動的または手動により発生するイベントを走査により検出する手段、

検出したイベントから、対応するプログラムを実行可能とする手段、

次に実行すべきプログラムを選択する手段、

中断したプログラムの中断点からの実行再開手段とを備えて、プログラムの並行実行を行うことを特徴とするプログラム並行実行装置。

3．発明の詳細な説明

［発明の技術分野］

この発明は一般的にシングルタスクOS（オペレーティングシステム）といわれているプログラムのもとで実行するコンピュータ装置において、プログラムの並行実行を行う監視プログラムに関するものである。また、本方式によるマルチタスクOSの構築に関するものである。

［発明の背景］

現在シングルタスクOSが広汎に使用されているが、そこで作成されるプログラムはその名の通り、直列に実行されるものである。また、見かけは並列であるが一方は他方が終了するまで実行できないものである。さらに、本OSを用いて並行的に動作させるものもあるが、ジョブ単位での切り替わりであり、限定されたイベントしか取り扱えない。これらは何れも、本発明のようにリアルタイムモニタの思想を組み込み、全イベントを有機的に組み合わせ並行に、かつ効率よくプログラムを動作させるものではない。

また、本発明より効率的なリアルタイム性を備えたモニタプログラムは数多くあるが独自の環境下に入るなど、リアルタイム性をそれほど必要としないプログラム開発には使いにくいものである。

［発明の目的］

それ故、この発明の目的は、ユーザの使いなれたシングルタスクＯＳコンピュータ装置内で、それほどリアルタイム性を必要としないが並行動作をさせたいときに、簡単に並行処理ーマルチタスク処理ーを構成できる装置を提供することにある。

〔発明の構成および効果〕

本発明は従来のリアルタイム型モニタの基本機能ープログラムの中断、イベントの検出、タスクスケジュール、プログラムの実行再開などーを有しているが、そのイベントの検出をシングルタスクＯＳ下でも可能なように走査ループとしたものである。

実行中のプログラムの中断は、本モニタに関する部分についてはタスクからの要求にもとづいて行われ、そのタスクの状態、つまりレジスターをそのタスクの専用エリアに退避することにより行われる。再開はそのタスクの専用エリアのデータを復元することにより行う。タスクの最初の実行開始はタスクの開始点を中断点とみなして行うのが簡単である。イベントの検出はデータが入力されたか、又出力されたかなど、各イベントに対応した調査を全イベントについて走査ループで調べて、タスクが次の処理を再開するための条件が発生していれば、そのタスクを実行可能状態とする。タスクスケジュールはタスクの管理テーブルを調べて、実行可能であるタスクを選択する。実行可能のタスクがいくつかある時は通常優先レベルの高いものを選択する。

また、あるタスクの処理が長くなる場合はタスクからの要求により、あるいはある一定時間の処理をした後に、本タスクの実行を中断して、他の優先度の高いタスクの処理を行わせることが出来る。

さらに、緊急を要するときは従来のリアルタイム方式、つまり、割り込み発生でイベントを検出して、割り込み処理で問題となる資源の排他制御を行った後に、処理が可能であれば、その割り込み処理を行うことによりリアルタイムでの応答も可能となる。なお、排他制御を必要としないときは、リアルタイム方式と同様に直ちに応答が出来る。

また、これにプログラムのローディング機能とジョブスケジュール機能を付加すればマルチジョブ・マルチタスクのシステムが構成できる。

効果としては

（１）　簡単にシングルタスクＯＳ下で並行処理が構築できる。並行処理により効率のよいプログラムとなる。また、並行処理でないと使いものにならないときもある。

（２）　通常の割り込み処理はシングルタスクＯＳが管理しているので並行処理にともなうプログラムバグが発生しにくい。

（３）　使いなれたＯＳ環境下なので、プログラム開発の向上がはかれる。

〔実施例の説明〕

この発明の一実施例を図面に基づいて説明する。説明は、まず第３図、４図のデータから行い、その後第１図、２図のプログラムについて行う。ここでは、並行処理を行うプログラムをすべてタスクと記述している。

第３図はタスク管理テーブルであり、１つのタスクについてタスクステータス１３とスタックポインタ１４及びその他各種データ１５があり、タスク単位に用意されている。ここではｎ個のタスクがあるものとする。なお本テーブルは優先レベルの高いものから順番に並べられている。タスクステータス１３はタスクの実行の可否の状態を記憶している。「０」は実行不可をあらわし、「１」は実行可とする。スタックポインタ１４はタスクの中断が発生したとき、レジスタ値をスタックにしまいこんだ後のスタックポインタの値を保存したものである。その他各種データ１５はタスクがタイムアップ時で中断したときのタイムカウント値、入出力完了待の時の入出力イベントとタスクの対応データやタスクの実行経過時間などがある。

第４図はスタックであり、各タスクに個別に割り当てられている。個数はタスク数と同様にｎ個である。なおこの図ではスタックの成長方向は上方としている。既使用エリア１８は中断が発生す

る直前までタスクで使用されていたエリアである。レジスタ退避エリア17は中断が発生したときに同タスクのスタックポインタを除く全レジスタを退避するエリアである。未使用エリア16は、スタックの残りの未使用エリアである。

第1図と第2図はプログラムのフローチャートである。第1図の処理1、2は次に実行すべきタスクを選択する部分である。ここでは、タスク管理テーブルのタスク1からタスクnまで順にタスクステータスを調べて、実行可であれば処理8へ分岐する。すべて実行不可であれば処理3へ移る。処理3～7がイベントを走査により検出して対応するイベントの処理を行う部分である。イベント数はm個あるものとする。イベントi（1≦i≦m）に対して、イベントiの発生処理ではタスクj（1≦j≦n）を実行可とする処理を含んでいる。実行可とはすなわち該当タスクのタスク管理テーブルのタスクステータスiを'1'にすることである。タスクjの選択は明示的に行うこともできるし、中断処理11で間接的に示されることもある。この情報はその他各種データ15に記憶されている。処理7は、イベントの発生処理でタスクを実行可とした場合は処理1へ分岐し、そうでないときは再び走査を行うために処理3へ分岐するための判断を行う部分である。処理8はタスクの中断点からの再開部分である。実行に移すタスクのタスク管理テーブルのスタックポインタ値をスタックポインタレジスタに復旧した後、スタックエリアから他のレジスタをプログラムカウンタを最後にして復旧する。この時点でタスクの実行が再開される。

第2図は中断をした時のフローチャートである。処理9ではスタックポインタを除く全レジスタをスタックエリアに退避する。再開にあわせて、プログラムカウンタを最初に退避しておく。処理10ではこの時点のスタックポインタレジスタをタスク管理テーブルの対応するタスクのスタックポインタ2へ退避する。処理11の中断処理はタスクからの要求によりさまざまであるが、時間待ち、入出力待ちや他のタスクの起動などがある。このとき、該当タスクがイベント待になるときは、対応するタスク管理テーブルのタスクステータス1を'0'にする。そして、必要なデータをその他各種データ3にセットする。この後、処理12により、第1図の処理を行うことになる。

4．図面の簡単な説明

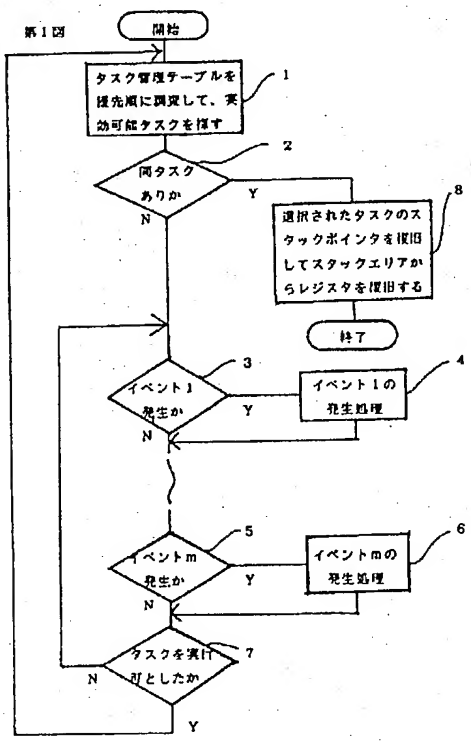
第1図は実行可能タスクの選択、イベントを検出するための走査部及び実行の再開部分のフローチャートである。

第2図は中断するときのフローチャートである。

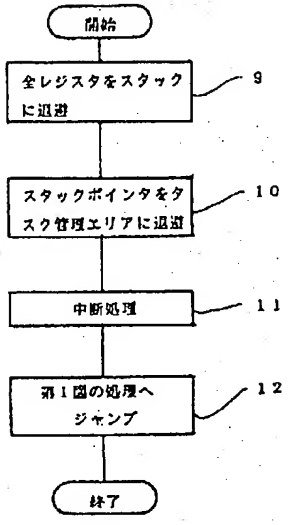
第3図はタスク管理テーブルである。

第4図はスタックエリアの構成を示したものである。

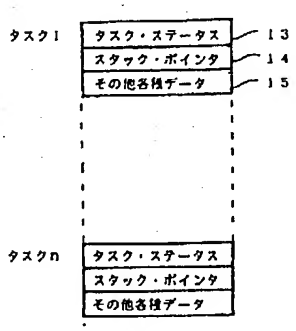
第1図



第2図




第3図




第4図